**Q4 . How functions works. Step by step :-**

Sure, let's break down how functions work in the context of a compiler. A compiler translates source code written in a high-level programming language into machine code or another lower-level representation. Here are the steps involved in the functioning of a compiler: 1. Lexical Analysis (Scanning): - The compiler scans the source code to identify and tokenize different components such as keywords, identifiers, literals, and symbols. - Tokens represent the basic building blocks of the programming language. 2. Syntax Analysis (Parsing): - The compiler parses the tokens to create a syntactic tree or abstract syntax tree (AST). - The tree structure represents the hierarchical relationships between different language constructs. - Functions and their parameters are identified during this phase. 3. Semantic Analysis: - The compiler performs semantic analysis to ensure that the program adheres to the language's rules and semantics. - It checks for correctness in variable usage, type compatibility, and other semantic rules. 4. Intermediate Code Generation: - The compiler generates an intermediate representation of the source code. - This representation is often an intermediate code, which is a platformindependent and high-level representation of the program. - Functions are typically identified, and intermediate code for function calls and definitions is generated. 5. Code Optimization: - The compiler performs various optimization techniques to improve the efficiency of the generated code. - This may include common subexpression elimination, dead code elimination, and loop optimization. 6. Code Generation: - The compiler translates the intermediate code into the target machine code or another lower-level representation. - This involves mapping the high-level constructs to machine instructions. - Functions are translated into the corresponding machine code, and memory locations for variables are determined. 7. Register Allocation: - The compiler assigns registers to variables to optimize the use of the processor's registers. - Register allocation is crucial for improving the performance of the generated code. 8. Code Linking and Assembly: - If the program consists of multiple source files or external libraries, the compiler may link them together. - The linked code is then assembled into an executable file. 9. Loader and Execution: - The loader loads the executable file into memory. - The program is executed by the computer's processor, following the instructions generated by the compiler. 10. Error Handling: - Throughout the compilation process, the compiler performs error checking and reports any syntax or semantic errors found in the source code. - Error messages are generated to help the programmer identify and fix issues. In summary, the functioning of a compiler involves multiple phases, each handling specific aspects of the compilation process. Functions in the source code are identified, and their corresponding machine code is generated during the intermediate code generation and code generation phases.